Apex Triggers

Get Started with Apex Trigger

AccountAddressTrigger

```
trigger AccountAddressTrigger on Account (before insert,
before update) {
for(Account account:Trigger.New){
if(account.Match_Billing_Address__c == True){
account.ShippingPostalCode = account.BillingPostalCode;
}
}
}
```

Bulk Apex Trigger

ClosedOpportunityTrigger

```
trigger ClosedOpportunityTrigger on Opportunity (after insert,
after update) {

List<Task> taskList = new List<Task>();

for(Opportunity opp : Trigger.New){

if(opp.StageName == 'Closed Won'){

taskList.add(new Task(Subject = 'Follow Up Test Task',WhatId
```

```
=opp.Id));
}
}
if(taskList.size() >0){
insert taskList;
}
}
```

Apex Testing

Get Started with Apex Unit Tests

VerifyDate

```
public class VerifyDate {
//method to handle potential checks against two dates
public static Date CheckDates(Date date1, Date date2) {
//if date2 is within the next 30 days of date1, use

date2. Otherwise use the end of the month
```

```
if(DateWithin30Days(date1,date2)) {
return date2;
} else {
return SetEndOfMonthDate(date1);
}
}
//method to check if date2 is within the next 30 days of
date1
private static Boolean DateWithin30Days(Date date1,
Date date2) {

//check for date2 being in the past

if( date2 &lt; date1) { return false; }
//check that date2 is within (&gt;=) 30 days of date1

Date date30Days = date1.addDays(30); //create a date
30 days away from date1

if( date2 &gt;= date30Days ) { return false; }
else { return true; }
}
//method to return the end of the month of a given date
private static Date SetEndOfMonthDate(Date date1) {
```

```
Integer totalDays = Date.daysInMonth(date1.year(),

date1.month());

Date lastDay = Date.newInstance(date1.year(),

date1.month(), totalDays);
return lastDay;
}
}
```

TestVerifyDate

```
@isTest

private class TestVerifyDate {

@isTest static void Test_CheckDates_case1(){

Date D = VerifyDate.CheckDates(date.parse('01/01/2020'),
date.parse('01/05/2020'));

System.assertEquals(date.parse('01/05/2020'), D);

}

@isTest static void Test_CheckDates_case2(){

Date D = VerifyDate.CheckDates(date.parse('01/01/2020'),
```

```
date.parse('01/05/2020'));

System.assertEquals(date.parse('01/31/2020'), D);

}

@isTest static void Test_DateWithin30Days_case1(){

Boolean flag =
VerifyDate.DateWithin30Days(date.parse('01/01/2020'),
date.parse('12/30/2019'));

System.assertEquals(false, flag);

}

@isTest static void Test_DateWithin30Days_case2(){

Boolean flag =
VerifyDate.DateWithin30Days(date.parse('01/01/2020'),
date.parse('02/02/2020'));

System.assertEquals(false, flag);

}

@isTest static void Test_DateWithin30Days_case3(){

Boolean flag =
VerifyDate.DateWithin30Days(date.parse('01/01/2020'),
```

```
date.parse('01/15/2020'));
System.assertEquals(true, flag);
}
@isTest static void Test_SetEndOfMonthDate(){
Date returndate =
VerifyDate.SetEndOfMonthDate(date.parse('01/01/2020'));
}
}
```

Test Apex Triggers

RestrictContactByName

```
trigger RestrictContactByName on Contact (before insert) {
//check contacts prior to insert or update for invalid data
For (Contact c : Trigger.New) {
if(c.LastName == 'INVALIDNAME') { //invalidname is invalid
c.AddError('The Last Name "'+c.LastName+'" is not allowed for
```

```
DML');
}
}
}
```

TestRestrictContactByName

```
@IsTest
public class TestRestrictContactByName {
@IsTest static void createBadContact(){
Contact c = new
Contact(FirstName='pavani',LastName='INVALIDNAME');
Test.startTest();
Database.SaveResult result = Database.insert(c, false);
Test.stopTest();
System.assert(!result.isSuccess());
}
```

}

Create Test Data for Apex Tests

RandomContactFactory

```
public class RandomContactFactory {

public static List<Contact> generateRandomContacts(Integer num, String lastName){

List<Contact>contactList = new List<Contact>();

for(Integer i = 1;i<=num;i++){

Contact ct = new Contact(FirstName = 'Test '+i, LastName =lastName);

contactList.add(ct);

}

return contactList;

}

}
```

Asynchronous Apex

Use Future Methods

AccountProcessor

```
public class AccountProcessor {

@future

public static void countContacts(List<Id> accountIds) {

List<Account> accountsToUpdate = new List<Account>();

List<Account> accounts = [Select Id, Name, (select Id from
Contacts) from Account Where Id IN :accountIds];

// process account records to do awesome stuff

For(Account acc:accounts){

List<Contact> contactList = acc.Contacts;

acc.Number_of_Contacts__c = contactList.size();

accountsToUpdate.add(acc);

}

update accountsToUpdate;
```

```
}
}
```

AccountProcessorTest

```
@IsTest
private class AccountProcessorTest {
@IsTest
private static void testCountContacts() {
Account newAccount = new Account(Name='Test Account');
insert newAccount;
Contact newContact1 = new Contact(FirstName='John',
LastName='Doe',
AccountId=newAccount.Id);
insert newContact1;
Contact newContact2 = new Contact(FirstName='Jane',
LastName='Doe',
```

```
AccountId=newAccount.Id);

insert newContact2;

accountIds.add(newAccount.Id);

Test.startTest();

AccountProcessor.countContacts(accountIds);

Test.stopTest();

}

}
```

Use Batch Apex

LeadProcessor

```
Global class LeadProcessor implements

Database.Batchable<sObject> {

global Database.QueryLocator start(Database.BatchableContext
bc) {

return Database.getQueryLocator(
```

```
'SELECT ID from Lead'

);

}

global void execute(Database.BatchableContext bc, List<Lead> scope){

// process each batch of records

List<Lead> leads = new List<Lead>();

for (Lead lead : scope) {

lead.LeadSource = 'Dreamforce';

leads.add(lead);

}

update leads;

}

Global void finish(Database.BatchableContext bc){

}

}
```

LeadProcessorTest

```
@isTest
private class LeadProcessorTest {
@testSetup
static void setup() {
List<Lead> leads = new List<Lead>();
// insert 10 accounts
for (Integer i=0;i<200;i++) {
leads.add(new Lead(LastName='Lead '+i,Company='Test Co'));
}
insert leads;
}
@isTest static void test() {
Test.startTest();
LeadProcessor myLeads = new LeadProcessor();
Id batchId = Database.executeBatch(myLeads);
```

```
Test.stopTest();

// after the testing stops, assert records were updated properly

System.assertEquals(200, [select count() from Lead where
LeadSource = 'Dreamforce']);

}

}
```

Control Process with Queueable Apex

AddPrimaryContact

```
public class AddPrimaryContact implements Queueable {

private Contact con;

private String state;

public AddPrimaryContact(Contact con, String state){

this.con = con;

this.state = state;

}
```

```
public void execute(QueueableContext context){

List<Account> accounts = [Select Id, Name, (Select FirstName,
LastName, Id from contacts)

from Account where BillingState = :state Limit 200];

List<Contact>primaryContacts = new List<Contact>();

for(Account acc:accounts){

Contact c = con.clone();

c.AccountId = acc.Id;

primaryContacts.add(c);

}

if(primarycontacts.size() > 0){

insert primaryContacts;

}

}

}
```

AddPrimaryContactTest

```
@isTest
public class AddPrimaryContactTest {
static testmethod void testQueueable(){
List<Account> testAccounts = new List<Account>();
for(Integer i=0;i<50;i++){
testAccounts.add(new Account(Name='Account
'+i,BillingState='CA'));
}
for(Integer j=0;j<50;j++){
testAccounts.add(NEW Account(Name='Account
'+j,BillingState='NY'));
}
insert testAccounts;
Contact testContact = new Contact(FirstName = 'John',
LastName ='Doe');
insert testContact;
AddPrimaryContact addit = new
```

```
addPrimaryContact(testContact, 'CA');

Test.startTest();

system.enqueueJob(addit);

Test.stopTest();

System.assertEquals(50,[Select count() from Contact where
AccountId in(Select Id from Account where
Billingstate='CA')]);

}

}
```

Schedule Jobs Using the Apex Scheduler

DailyLeadProcessor

```
public without sharing class DailyLeadProcessor implements
Schedulable{

public void execute(SchedulableContext ctx){

List<Lead> leads = [SELECT Id, LeadSource FROM Lead
WHERE LeadSource = null LIMIT 200];
```

```
for(Lead l : leads){
l.LeadSource = 'Dreamforce';
}
update leads;
}
}
```

DailyLeadProcessorTest

```
@isTest
public class DailyLeadProcessorTest {
private static String CRON_EXP = '0 0 0 ? * * *';
@isTest
private static void testScheduledJob(){
List<Lead> leads = new List<lead>();
for (Integer i=0; i<500; i++){
if(i<250){
leads.add(new Lead(LastName='Connock',
```

```
Company='Salesforce'));

}else{

leads.add(new Lead(LastName='Connock',
Company='Salesforce', LeadSource='Other'));

}

}

insert leads;

Test.startTest();

String jobId = System.schedule('Process Leads', CRON_EXP,
new DailyLeadProcessor());

Test.stopTest();

List<Lead> updatedleads = [Select Id, LeadSource FROM Lead
WHERE LeadSource = 'Dreamforce'];

System.assertEquals(200, updatedLeads.size(),'ERROR:At least
1 record not updated correctly');

List<CronTrigger> cts = [SELECT Id, TimesTriggered,
NextFireTime FROM CronTrigger WHERE Id = :jobId];

System.debug('Next Fire Time' + cts[0].NextFireTime);
```

```
}

}
```

Apex Integration Services

Apex REST Callouts

AnimalCallouts

```
public class AnimalsCallouts {

public static HttpResponse makeGetCallout() {

Http http = new Http();

HttpRequest request = new HttpRequest();

request.setEndpoint('https://th-apex-http-
callout.herokuapp.com/animals');

request.setMethod('GET');

HttpResponse response = http.send(request);

// If the request is successful, parse the JSON response.

if(response.getStatusCode() == 200) {

// Deserializes the JSON string into collections of primitive data
```

types.

```
Map<String, Object> results = (Map<String, Object>)
JSON.deserializeUntyped(response.getBody());

// Cast the values in the 'animals' key as a list

List<Object> animals = (List<Object>) results.get('animals');

System.debug('Received the following animals:');

for(Object animal: animals) {

System.debug(animal);

}

}

return response;

}

public static HttpResponse makePostCallout() {

Http http = new Http();

HttpRequest request = new HttpRequest();

request.setEndpoint('https://th-apex-http-
callout.herokuapp.com/animals');
```

```
request.setMethod('POST');
request.setHeader('Content-Type',
'application/json;charset=UTF-8');
request.setBody('{"name":"mighty moose"}');
HttpResponse response = http.send(request);
if(response.getStatusCode() != 201) {
System.debug('The status code returned was not expected: ' +
response.getStatusCode() + ' ' + response.getStatus());
} else {
System.debug(response.getBody());
}
return response;
}
}
```

AnimalCalloutsTest

```
@isTest
private class AnimalsCalloutsTest {
@isTest static void testGetCallout() {
// Create the mock response based on a static resource
StaticResourceCalloutMock mock = new
StaticResourceCalloutMock();
mock.setStaticResource('GetAnimalResource');
mock.setStatusCode(200);
mock.setHeader('Content-Type', 'application/json;charset=UTF-
8');
// Associate the callout with a mock response
Test.setMock(HttpCalloutMock.class, mock);
// Call method to test
HttpResponse result = AnimalsCallouts.makeGetCallout();
// Verify mock response is not null
System.assertNotEquals(null,result, 'The callout returned a null
response.');
```

```
// Verify status code
System.assertEquals(200,result.getStatusCode(), 'The status code is not 200.');
// Verify content type
System.assertEquals('application/json;charset=UTF-8',
result.getHeader('Content-Type'),
'The content type value is not expected.');
// Verify the array contains 3 items
Map<String, Object> results = (Map<String, Object>)
JSON.deserializeUntyped(result.getBody());
List<Object> animals = (List<Object>) results.get('animals');
System.assertEquals(3, animals.size(), 'The array should only contain 3 items.');
}
@isTest
static void testPostCallout() {
// Set mock callout class
```

```
Test.setMock(HttpCalloutMock.class, new
AnimalsHttpCalloutMock());

// This causes a fake response to be sent

// from the class that implements HttpCalloutMock.

HttpResponse response = AnimalsCallouts.makePostCallout();

// Verify that the response received contains fake values

String contentType = response.getHeader('Content-Type');

System.assert(contentType == 'application/json');

String actualValue = response.getBody();

System.debug(response.getBody());

String expectedValue = '{"animals": ["majestic badger", "fluffy
bunny", "scary bear", "chicken", "mighty moose"]}';

System.assertEquals(expectedValue, actualValue);

System.assertEquals(200, response.getStatusCode());

}

}
```

AnimalsHttpCalloutMock

```
@isTest
global class AnimalsHttpCalloutMock implements HttpCalloutMock {
// Implement this interface method
global HTTPResponse respond(HTTPRequest request) {
// Create a fake response
HttpResponse response = new HttpResponse();
response.setHeader('Content-Type', 'application/json');
response.setBody('{"animals": ["majestic badger", "fluffy bunny", "scary bear", "chicken", "mighty moose"]}');
response.setStatusCode(200);
return response;
}
}
```

AnimalLocator

```
public class AnimalLocator {

public static String getAnimalNameById(Integer i) {

String animalName;

Http http = new Http();

HttpRequest request = new HttpRequest();

request.setEndpoint('https://th-apex-http-
callout.herokuapp.com/animals/'+i);

request.setMethod('GET');

HttpResponse response = http.send(request);

// If the request is successful, parse the JSON response.

Map<String, Object> result = (Map<String, Object>)
JSON.deserializeUntyped(response.getBody());

Map<String, Object>animal = (Map<String,
Object>)result.get('animal');

return string.valueOf(animal.get('name'));

}

}
```

AnimalLocatorTest

```
@isTest
private class AnimalLocatorTest {
@isTest
static void AnimalLocatorTest1() {
Test.setMock(HttpCalloutMock.class, new
AnimalLocatorMock());
String actual = AnimalLocator.getAnimalNameById(1);
String expected = 'moose';
System.assertEquals(actual,expected);
}
}
```

AnimalLocatorMock

```
@isTest
global class AnimalLocatorMock implements HttpCalloutMock
{
```

```
global HTTPResponse respond(HTTPRequest request) {

HttpResponse response = new HttpResponse();

response.setHeader('Content-Type', 'application/json');

response.setBody('{"animals": {"id":1,
"name":"chicken","eats":"chicken food","says":"cluck
cluck"}}');

response.setStatusCode(200);

return response;

}

}
```

Apex SOAP Callouts

AwesomeCalculator

```
public class AwesomeCalculator {
public static Double add(Double x, Double y) {
calculatorServices.CalculatorImplPort calculator =
new calculatorServices.CalculatorImplPort();
```

```
return calculator.doAdd(x,y);
}
}
```

CalculatorCalloutMock

```
@isTest
global class CalculatorCalloutMock implements
WebServiceMock {
global void doInvoke(
Object stub,
Object request,
Map<String, Object> response,
String endpoint,
String soapAction,
String requestName,
String responseNS,
String responseName,
String responseType) {
// start - specify the response you want to send
calculatorServices.doAddResponse response_x =
new calculatorServices.doAddResponse();
response_x.return_x = 3.0;
// end

response.put(response_x',& response_x);
```

}
}

AwesomeCalculatorTest

```
@isTest
private class AwesomeCalculatorTest {
@isTest static void testCallout() {
// This causes a fake response to be generated
Test.setMock(WebServiceMock.class, new
CalculatorCalloutMock());
// Call the method that invokes a callout
Double x = 1.0;
Double y = 2.0;
Double result = AwesomeCalculator.add(x, y);
// Verify that a fake result is returned
System.assertEquals(3.0, result);
}
}
```

ParkLocator

```
public class ParkLocator {
public static List &lt; String &gt; country(String country) {

ParkService.ParksImplPort prkSvc = new
ParkService.ParksImplPort();
return prkSvc.byCountry(country);
```

```
}
}
```

ParkLocatorTest

```
@isTest

private class ParkLocatorTest {

@isTest static void testCallout () {

Test.setMock(WebServiceMock.class, new ParkServiceMock());

String country = 'United States';

List<String> expectedParks = new List<String>{'Yosemite',
'Sequoia', 'Crater Lake'};

System.assertEquals(expectedParks,ParkLocator.country(countr
y));

}

}
```

ParkService

```
//Generated by wsdl2apex
```

```
public class ParkService {

public class byCountryResponse {

public String[] return_x;

private String[] return_x_type_info = new
String[]{'return','http://parks.services/',null,'0','-1','false'};

private String[] apex_schema_type_info = new
String[]{'http://parks.services/','false','false'};

private String[] field_order_type_info = new
String[]{'return_x'};

}

public class byCountry {

public String arg0;

private String[] arg0_type_info = new
String[]{'arg0','http://parks.services/',null,'0','1','false'};

private String[] apex_schema_type_info = new
String[]{'http://parks.services/','false','false'};

private String[] field_order_type_info = new String[]{'arg0'};

}
```

```
public class ParksImplPort {

public String endpoint_x = 'https://th-apex-soap-
service.herokuapp.com/service/parks';

public Map<String,String> outputHttpHeaders_x;

public String clientCertName_x;

public String clientCert_x;

public String clientCertPasswd_x;

public Integer timeout_x;

private String[] ns_map_type_info = new
String[]{'http://parks.services/', 'ParkService'};

public String[] byCountry(String arg0) {

ParkService.byCountry request_x = new
ParkService.byCountry();

request_x.arg0 = arg0;

ParkService.byCountryResponse response_x;

Map<String, ParkService.byCountryResponse>
response_map_x = new Map<String,
ParkService.byCountryResponse>();
```

```
response_map_x.put('response_x', response_x);
WebServiceCallout.invoke(
this,
request_x,
response_map_x,
new String[]{endpoint_x,
'',
'http://parks.services/',
'byCountry',
'http://parks.services/',
'byCountryResponse',
'ParkService.byCountryResponse'}
);
response_x = response_map_x.get('response_x');
return response_x.return_x;
}
```

```
}

}
```

ParkServiceMock

```
@isTest

global class ParkServiceMock implements WebServiceMock {

global void doInvoke(

Object stub,

Object request,

Map<String, Object> response,

String endpoint,

String soapAction,

String requestName,

String responseNS,

String responseName,

String responseType) {
```

```
// start - specify the response you want to send

parkService.byCountryResponse response_x =

new parkService.byCountryResponse();

response_x.return_x = new List<String>{'Yosemite', 'Sequoia',
'Crater Lake'};

// end

response.put('response_x', response_x);

}

}
```

AsyncParkService

```
//Generated by wsdl2apex

public class AsyncParkService {

public class byCountryResponseFuture extends
System.WebServiceCalloutFuture {

public String[] getValue() {

ParkService.byCountryResponse response =
```

```
(ParkService.byCountryResponse)System.WebServiceCallout.endInvoke(this);

return response.return_x;

}

}

public class AsyncParksImplPort {

public String endpoint_x = 'https://th-apex-soap-service.herokuapp.com/service/parks';

public Map<String,String> inputHttpHeaders_x;

public String clientCertName_x;

public Integer timeout_x;

private String[] ns_map_type_info = new String[]{'http://parks.services/', 'ParkService'};

public AsyncParkService.byCountryResponseFuture beginByCountry(System.Continuation continuation,String arg0) {

ParkService.byCountry request_x = new ParkService.byCountry();
```

```
request_x.arg0 = arg0;
return (AsyncParkService.byCountryResponseFuture)
System.WebServiceCallout.beginInvoke(
this,
request_x,
AsyncParkService.byCountryResponseFuture.class,
continuation,
new String[]{endpoint_x,
'',
'http://parks.services/',
'byCountry',
'http://parks.services/',
'byCountryResponse',
'ParkService.byCountryResponse'}
);
}
```

```
}

}
```

Apex Web Services

CaseManager

```
@RestResource(urlMapping='/Cases/*')
global with sharing class CaseManager {
@HttpGet
global static Case getCaseById() {
RestRequest request = RestContext.request;
// grab the caseId from the end of the URL
String caseId = request.requestURI.substring(
request.requestURI.lastIndexOf('/';)+1);
Case result = [SELECT
CaseNumber,Subject,Status,Origin,Priority
FROM Case
WHERE Id = :caseId];
return result;

}
```

```
@HttpPost
global static ID createCase(String subject, String status,
String origin, String priority) {
Case thisCase = new Case(
Subject=subject,
Status=status,
Origin=origin,
Priority=priority);
insert thisCase;
return thisCase.Id;
}
@HttpDelete
global static void deleteCase() {
RestRequest request = RestContext.request;
String caseId = request.requestURI.substring(
request.requestURI.lastIndexOf('/')+1);
Case thisCase = [SELECT Id FROM Case WHERE Id =
:caseId];
delete thisCase;
}
@HttpPut

global static ID upsertCase(String subject, String status,
String origin, String priority, String id) {
```

```
Case thisCase = new Case(
Id=id,
Subject=subject,
Status=status,
Origin=origin,
Priority=priority);
// Match case by Id, if present.
// Otherwise, create new case.
upsert thisCase;
// Return the case ID.
return thisCase.Id;
}
@HttpPatch
global static ID updateCaseFields() {
RestRequest request = RestContext.request;
String caseId = request.requestURI.substring(
request.requestURI.lastIndexOf('/')+1);
Case thisCase = [SELECT Id FROM Case WHERE Id =
:caseId];
// Deserialize the JSON string into name-value pairs

Map&lt;String, Object&gt; params = (Map&lt;String,
Object&gt;)JSON.deserializeUntyped(request.requestbody.
tostri
```

```
ng());
// Iterate through each parameter field and value
for(String fieldName : params.keySet()) {
// Set the field and value on the Case sObject
thisCase.put(fieldName, params.get(fieldName));
}
update thisCase;
return thisCase.Id;
}
}
```

CaseManagerTest

```
@IsTest
private class CaseManagerTest {
@isTest static void testGetCaseById() {
Id recordId = createTestRecord();
// Set up a test request
RestRequest request = new RestRequest();
request.requestUri =
&#39;https://yourInstance.my.salesforce.com/services/ap
exrest/C
ases/'

+ recordId;
request.httpMethod = 'GET'
```

```
RestContext.request = request;
// Call the method to test
Case thisCase = CaseManager.getCaseById();
// Verify results
System.assert(thisCase != null);
System.assertEquals('Test record', thisCase.Subject);
}
@isTest static void testCreateCase() {
// Call the method to test
ID thisCaseId = CaseManager.createCase(
&#39;Ferocious chipmunk','New', 'Phone', 'Low');
// Verify results
System.assert(thisCaseId != null);
Case thisCase = [SELECT Id,Subject FROM Case WHERE
Id=:thisCaseId];
System.assert(thisCase != null);
System.assertEquals(thisCase.Subject, 'Ferocious
chipmunk');
}
@isTest static void testDeleteCase() {

Id recordId = createTestRecord();
// Set up a test request
RestRequest request = new RestRequest();
```

```
request.requestUri =
'https://yourInstance.my.salesforce.com/services/apexrest/C
ases/'
+ recordId;
request.httpMethod = 'DELETE'
RestContext.request = request;
// Call the method to test
CaseManager.deleteCase();
// Verify record is deleted
List&lt;Case&gt; cases = [SELECT Id FROM Case WHERE
Id=:recordId];
System.assert(cases.size() == 0);
}
@isTest static void testUpsertCase() {
// 1. Insert new record
ID case1Id = CaseManager.upsertCase(
'Ferocious chipmunk', 'New', 'Phone', 'Low', null);
// Verify new record was created
System.assert(Case1Id != null);

Case case1 = [SELECT Id,Subject FROM Case WHERE
Id=:case1Id];
System.assert(case1 != null);
```

```
System.assertEquals(case1.Subject, 'Ferocious
chipmunk');
// 2. Update status of existing record to Working
ID case2Id = CaseManager.upsertCase(
'Ferocious chipmunk', 'Working', 'Phone', 'Low',
case1Id);
// Verify record was updated
System.assertEquals(case1Id, case2Id);
Case case2 = [SELECT Id,Status FROM Case WHERE
Id=:case2Id];
System.assert(case2 != null);
System.assertEquals(case2.Status, 'Working');
}
@isTest static void testUpdateCaseFields() {
Id recordId = createTestRecord();
RestRequest request = new RestRequest();
request.requestUri =
'https://yourInstance.my.salesforce.com/services/apexres
t/C
ases/';
+ recordId;

request.httpMethod = 'PATCH';
request.addHeader('Content-Type';, 'application/json');
```

```
request.requestBody = Blob.valueOf('{&quot;status&quot;:
&quot;Working&quot;}');
RestContext.request = request;
// Update status of existing record to Working
ID thisCaseId = CaseManager.updateCaseFields();
// Verify record was updated
System.assert(thisCaseId != null);
Case thisCase = [SELECT Id,Status FROM Case WHERE
Id=:thisCaseId];
System.assert(thisCase != null);
System.assertEquals(thisCase.Status,'Working');
}
// Helper method
static Id createTestRecord() {
// Create test record
Case caseTest = new Case(
Subject='Test record';,
Status='New',
Origin='Phone',
Priority='Medium');

insert caseTest;
return caseTest.Id;
}
```

```
}
```

AccountManager

```
@RestResource(urlMapping='/Accounts/*/contacts')
global with sharing class AccountManager {
@HttpGet
global static Account getAccount(){

RestRequest request = RestContext.request;
String accountId =
request.requestURI.substringBetween('Accounts/','/contac
ts'
);
Account result = [SELECT Id, Name, (Select Id, Name from
Contacts) from Account where Id=:accountId];
return result;
}
}
```

AccountManagerTest

```
@isTest
private class AccountManagerTest {
@isTest
static void testGetAccount(){
Account a = new Account(Name='TestAccount');
insert a;
```

```
Contact c = new Contact(AccountId=a.Id,
FirstName='Test', LastName='Test');
insert c;
RestRequest request = new RestRequest();
request.requestUri =
'https://yourInstance.my.salesforce.com/services/apexres
t//
Accounts/'+a.id+'/contacts';

request.httpMethod = 'GET';
RestContext.request = request;
Account myAcct = AccountManager.getAccount();
System.assert(myAcct != null);
System.assertEquals('TestAccount', myAcct.Name);
}
}
```

APEX SPECIALIST SUPERBADGE

Challenge 2-Automate Record Creation

MaintenanceRequestHelper

```
public with sharing class MaintenanceRequestHelper {
public static void updateworkOrders(List<Case>
```

```
updWorkOrders, Map<Id,Case> nonUpdCaseMap) {
Set<Id> validIds = new Set<Id>();
For (Case c : updWorkOrders){
if (nonUpdCaseMap.get(c.Id).Status != 'Closed' &&
c.Status == 'Closed'){
if (c.Type == 'Repair' || c.Type == 'Routine
Maintenance'){
validIds.add(c.Id);
}
}
}
if (!validIds.isEmpty()){
List<Case> newCases = new List<Case>();
Map<Id,Case> closedCasesM = new
Map<Id,Case>([SELECT
Id, Vehicle__c, Equipment__c,
Equipment__r.Maintenance_Cycle__c,(SELECT

Id,Equipment__c,Quantity__c FROM
Equipment_Maintenance_Items__r)
FROM Case WHERE Id IN
:validIds]);
Map<Id,Decimal> maintenanceCycles = new
Map<ID,Decimal>();
```

```
AggregateResult[] results = [SELECT
Maintenance_Request__c,
MIN(Equipment__r.Maintenance_Cycle__c)cycle FROM
Equipment_Maintenance_Item__c WHERE
Maintenance_Request__c IN :ValidIds GROUP BY
Maintenance_Request__c];
for (AggregateResult ar : results){
maintenanceCycles.put((Id)
ar.get('Maintenance_Request__c'), (Decimal)
ar.get('cycle'));
}
for(Case cc : closedCasesM.values()){
Case nc = new Case (
ParentId = cc.Id,
Status = 'New',
Subject = 'Routine Maintenance',
Type = 'Routine Maintenance',
Vehicle__c = cc.Vehicle__c,
Equipment__c =cc.Equipment__c,

Origin = 'Web',
Date_Reported__c = Date.Today()
);
If (maintenanceCycles.containskey(cc.Id)){
```

```
nc.Date_Due__c = Date.today().addDays((Integer)
maintenanceCycles.get(cc.Id));
}
newCases.add(nc);
}
insert newCases;
List<Equipment_Maintenance_Item__c> clonedWPs =
new List<Equipment_Maintenance_Item__c>();
for (Case nc : newCases){
for (Equipment_Maintenance_Item__c wp :
closedCasesM.get(nc.ParentId).Equipment_Maintenance_I
te
ms__r){
Equipment_Maintenance_Item__c wpClone =
wp.clone();
wpClone.Maintenance_Request__c = nc.Id;
ClonedWPs.add(wpClone);
}
}
insert ClonedWPs;

}
}
}
```

MaintenanceRequest

```
trigger MaintenanceRequest on Case (before update, after
update) {
if(Trigger.isUpdate && Trigger.isAfter){
MaintenanceRequestHelper.updateWorkOrders(Trigger.Ne
w,
Trigger.OldMap);
}
}
```

Challenge 3- Synchronize Salesforce dat a with an external system

WarehouseCalloutServiece

```
public with sharing class WarehouseCalloutService
implements Queueable {

private static final String WAREHOUSE_URL = 'https://th-
superbadge-apex.herokuapp.com/equipment';

//class that makes a REST callout to an external
warehouse
```

system to get a list of equipment that needs to be updated.
//The callout’s JSON response returns the equipment records that you upsert in Salesforce.

```
@future(callout=true)
public static void runWarehouseEquipmentSync(){
Http http = new Http();
HttpRequest request = new HttpRequest();
request.setEndpoint(WAREHOUSE_URL);
request.setMethod('GET');
HttpResponse response = http.send(request);
List<Product2> warehouseEq = new List<Product2>();
if (response.getStatusCode() == 200){
List<Object> jsonResponse =
(List<Object>)JSON.deserializeUntyped(response.getBody
());
System.debug(response.getBody());
//class maps the following fields: replacement part
(always true), cost, current inventory, lifespan,
maintenance
cycle, and warehouse SKU
//warehouse SKU will be external ID for identifying
which equipment records to update within Salesforce
```

```
for (Object eq : jsonResponse){
Map<String,Object> mapJson =
(Map<String,Object>)eq;
Product2 myEq = new Product2();
myEq.Replacement_Part__c = (Boolean)
mapJson.get('replacement');

myEq.Name = (String) mapJson.get('name');
myEq.Maintenance_Cycle__c = (Integer)
mapJson.get('maintenanceperiod');
myEq.Lifespan_Months__c = (Integer)
mapJson.get('lifespan');
myEq.Cost__c = (Integer) mapJson.get('cost');
myEq.Warehouse_SKU__c = (String)
mapJson.get('sku');
myEq.Current_Inventory__c = (Double)
mapJson.get('quantity');
myEq.ProductCode = (String) mapJson.get('_id');
warehouseEq.add(myEq);
}
if (warehouseEq.size() > 0){
upsert warehouseEq;
System.debug('Your equipment was synced with the
warehouse one');
```

```
}
}
}
public static void execute (QueueableContext context){
runWarehouseEquipmentSync();
}

}
```

Challenge 4-Schedule synchronization using Apex code

WarehouseSyncShedule

```
global with sharing class WarehouseSyncSchedule
implements Schedulable{
global void execute(SchedulableContext ctx){
System.enqueueJob(new WarehouseCalloutService());
}
}
```

Challenge 5- Test automation logic

MaintenanceRequestHelperTest

```
@istest
public with sharing class MaintenanceRequestHelperTest
{
private static final string STATUS_NEW = 'New';
```

```
private static final string WORKING = 'Working';
private static final string CLOSED = 'Closed';
private static final string REPAIR = 'Repair';

private static final string REQUEST_ORIGIN = 'Web';
private static final string REQUEST_TYPE = 'Routine
Maintenance';
private static final string REQUEST_SUBJECT = 'Testing
subject';
PRIVATE STATIC Vehicle__c createVehicle(){
Vehicle__c Vehicle = new Vehicle__C(name =
'SuperTruck');
return Vehicle;
}
PRIVATE STATIC Product2 createEq(){
product2 equipment = new product2(name =
'SuperEquipment',
lifespan_months__C = 10,
maintenance_cycle__C = 10,
replacement_part__c = true);
return equipment;
}
PRIVATE STATIC Case createMaintenanceRequest(id
vehicleId, id equipmentId){
```

```
case cs = new case(Type=REPAIR,
Status=STATUS_NEW,
Origin=REQUEST_ORIGIN,

Subject=REQUEST_SUBJECT,
Equipment__c=equipmentId,
Vehicle__c=vehicleId);
return cs;
}
PRIVATE STATIC Equipment_Maintenance_Item__c
createWorkPart(id equipmentId,id requestId){
Equipment_Maintenance_Item__c wp = new
Equipment_Maintenance_Item__c(Equipment__c =
equipmentId,
Maintenance_Request__c = requestId);
return wp;
}

@istest
private static void testMaintenanceRequestPositive(){
Vehicle__c vehicle = createVehicle();
insert vehicle;
id vehicleId = vehicle.Id;
Product2 equipment = createEq();
```

```
insert equipment;
id equipmentId = equipment.Id;

case somethingToUpdate =
createMaintenanceRequest(vehicleId,equipmentId);
insert somethingToUpdate;
Equipment_Maintenance_Item__c workP =
createWorkPart(equipmentId,somethingToUpdate.id);
insert workP;
test.startTest();
somethingToUpdate.status = CLOSED;
update somethingToUpdate;
test.stopTest();
Case newReq = [Select id, subject, type, Equipment__c,
Date_Reported__c, Vehicle__c, Date_Due__c
from case
where status =:STATUS_NEW];
Equipment_Maintenance_Item__c workPart =
[select id
from
Equipment_Maintenance_Item__c
where Maintenance_Request__c
=:newReq.Id];
system.assert(workPart != null);
```

```
system.assert(newReq.Subject != null);
system.assertEquals(newReq.Type, REQUEST_TYPE);

SYSTEM.assertEquals(newReq.Equipment__c,
equipmentId);
SYSTEM.assertEquals(newReq.Vehicle__c, vehicleId);
SYSTEM.assertEquals(newReq.Date_Reported__c,
system.today());
}
@istest
private static void testMaintenanceRequestNegative(){
Vehicle__C vehicle = createVehicle();
insert vehicle;
id vehicleId = vehicle.Id;
product2 equipment = createEq();
insert equipment;
id equipmentId = equipment.Id;
case emptyReq =
createMaintenanceRequest(vehicleId,equipmentId);
insert emptyReq;
Equipment_Maintenance_Item__c workP =
createWorkPart(equipmentId, emptyReq.Id);
insert workP;
test.startTest();
```

```
emptyReq.Status = WORKING;
update emptyReq;

test.stopTest();
list<case> allRequest = [select id
from case];
Equipment_Maintenance_Item__c workPart =
[select id
from
Equipment_Maintenance_Item__c
where Maintenance_Request__c
= :emptyReq.Id];

system.assert(workPart != null);
system.assert(allRequest.size() == 1);
}
@istest
private static void testMaintenanceRequestBulk(){
list<Vehicle__C> vehicleList = new list<Vehicle__C>();
list<Product2> equipmentList = new list<Product2>();
list<Equipment_Maintenance_Item__c> workPartList =
new list<Equipment_Maintenance_Item__c>();
list<case> requestList = new list<case>();
list<id> oldRequestIds = new list<id>();
```

```
for(integer i = 0; i < 300; i++){
vehicleList.add(createVehicle());

equipmentList.add(createEq());
}
insert vehicleList;
insert equipmentList;
for(integer i = 0; i < 300; i++){

requestList.add(createMaintenanceRequest(vehicleList.ge
t(i).
id, equipmentList.get(i).id));
}
insert requestList;
for(integer i = 0; i < 300; i++){
workPartList.add(createWorkPart(equipmentList.get(i).id,
requestList.get(i).id));
}
insert workPartList;
test.startTest();
for(case req : requestList){
req.Status = CLOSED;
oldRequestIds.add(req.Id);
}
```

```
update requestList;
test.stopTest();
list<case> allRequests = [select id

from case
where status =: STATUS_NEW];

list<Equipment_Maintenance_Item__c> workParts =
[select id
from
Equipment_Maintenance_Item__c
where
Maintenance_Request__c in: oldRequestIds];

system.assert(allRequests.size() == 300);
}
}
```

MaintenanceRequestHelper

```
public with sharing class MaintenanceRequestHelper {
public static void updateworkOrders(List<Case>
updWorkOrders, Map<Id,Case> nonUpdCaseMap) {
Set<Id> validIds = new Set<Id>();

For (Case c : updWorkOrders){
```

```
if (nonUpdCaseMap.get(c.Id).Status != 'Closed' &&
c.Status == 'Closed'){

if (c.Type == 'Repair' || c.Type == 'Routine
Maintenance'){
validIds.add(c.Id);
}
}
}
if (!validIds.isEmpty()){
List<Case> newCases = new List<Case>();
Map<Id,Case> closedCasesM = new
Map<Id,Case>([SELECT Id, Vehicle__c, Equipment__c,
Equipment__r.Maintenance_Cycle__c,(SELECT
Id,Equipment__c,Quantity__c FROM
Equipment_Maintenance_Items__r)
FROM Case WHERE Id IN
:validIds]);
Map<Id,Decimal> maintenanceCycles = new
Map<ID,Decimal>();
AggregateResult[] results = [SELECT
Maintenance_Request__c,
MIN(Equipment__r.Maintenance_Cycle__c)cycle FROM
Equipment_Maintenance_Item__c WHERE
```

```
Maintenance_Request__c IN :ValidIds GROUP BY
Maintenance_Request__c];
for (AggregateResult ar : results){

maintenanceCycles.put((Id)
ar.get('Maintenance_Request__c'), (Decimal)
ar.get('cycle'));
}
for(Case cc : closedCasesM.values()){
Case nc = new Case (
ParentId = cc.Id,
Status = 'New',
Subject = 'Routine Maintenance',
Type = 'Routine Maintenance',
Vehicle__c = cc.Vehicle__c,
Equipment__c =cc.Equipment__c,
Origin = 'Web',
Date_Reported__c = Date.Today()
);
If (maintenanceCycles.containskey(cc.Id)){
nc.Date_Due__c = Date.today().addDays((Integer)
maintenanceCycles.get(cc.Id));
}
newCases.add(nc);
```

```
}
insert newCases;
List<Equipment_Maintenance_Item__c> clonedWPs =
new List<Equipment_Maintenance_Item__c>();

for (Case nc : newCases){
for (Equipment_Maintenance_Item__c wp :
closedCasesM.get(nc.ParentId).Equipment_Maintenance_I
te
ms__r){
Equipment_Maintenance_Item__c wpClone =
wp.clone();
wpClone.Maintenance_Request__c = nc.Id;
ClonedWPs.add(wpClone);
}
}
insert ClonedWPs;
}
}
}
```

MaintenanceRequest

```
trigger MaintenanceRequest on Case (before update, after
update) {
if(Trigger.isUpdate && Trigger.isAfter){
```

```
MaintenanceRequestHelper.updateWorkOrders(Trigger.New,
Trigger.OldMap);
}
}
```

Challenge 6- Test callout logic

WarehouseCalloutService

```
public with sharing class WarehouseCalloutService implements Queueable {
private static final String WAREHOUSE_URL =

'https://th-superbadge-apex.herokuapp.com/equipment';

//class that makes a REST callout to an external warehouse system to get a list of equipment that needs to be updated.
//The callout’s JSON response returns the equipment records that you upsert in Salesforce.
@future(callout=true)
public static void runWarehouseEquipmentSync(){
Http http = new Http();
HttpRequest request = new HttpRequest();
request.setEndpoint(WAREHOUSE_URL);
```

```
request.setMethod('GET');
HttpResponse response = http.send(request);
List<Product2> warehouseEq = new
List<Product2>();
if (response.getStatusCode() == 200){

List<Object> jsonResponse =
(List<Object>)JSON.deserializeUntyped(response.getBo
dy());
System.debug(response.getBody());
//class maps the following fields:
replacement part (always true), cost, current
inventory, lifespan, maintenance cycle, and warehouse
SKU
//warehouse SKU will be external ID for
identifying which equipment records to update within
Salesforce
for (Object eq : jsonResponse){
Map<String,Object> mapJson =
(Map<String,Object>)eq;
Product2 myEq = new Product2();
myEq.Replacement_Part__c = (Boolean)
mapJson.get('replacement');
myEq.Name = (String) mapJson.get('name');
```

```
myEq.Maintenance_Cycle__c = (Integer)
mapJson.get('maintenanceperiod');
myEq.Lifespan_Months__c = (Integer)
mapJson.get('lifespan');
myEq.Cost__c = (Integer) mapJson.get('cost');

myEq.Warehouse_SKU__c = (String)
mapJson.get('sku');
myEq.Current_Inventory__c = (Double)
mapJson.get('quantity');
myEq.ProductCode = (String)
mapJson.get('_id');
warehouseEq.add(myEq);
}
if (warehouseEq.size() > 0){
upsert warehouseEq;
System.debug('Your equipment was synced
with the warehouse one');
}
}
}
public static void execute (QueueableContext
context){
runWarehouseEquipmentSync();
```

```
}
}
```

WarehouseCalloutServiceTest

```
@IsTest
private class WarehouseCalloutServiceTest {
// implement your mock callout test here
@isTest
static void testWarehouseCallout() {
test.startTest();
test.setMock(HttpCalloutMock.class, new
WarehouseCalloutServiceMock());
WarehouseCalloutService.execute(null);
test.stopTest();
List<Product2> product2List = new List<Product2>();
product2List = [SELECT ProductCode FROM Product2];
System.assertEquals(3, product2List.size());
System.assertEquals('55d66226726b611100aaf741',
product2List.get(0).ProductCode);
System.assertEquals('55d66226726b611100aaf742',
product2List.get(1).ProductCode);
System.assertEquals('55d66226726b611100aaf743',
product2List.get(2).ProductCode);
}
}
```

WarehouseCalloutServiceMock

```
@isTest
global class WarehouseCalloutServiceMock implements
HttpCalloutMock {
// implement http mock callout
global static HttpResponse respond(HttpRequest request)
{
HttpResponse response = new HttpResponse();
response.setHeader('Content-Type', 'application/json');
response.setBody('[{"_id":"55d66226726b611100aaf741","
re
placement":false,"quantity":5,"name":"Generator 1000
kW","maintenanceperiod":365,"lifespan":120,"cost":5000,"s
ku":"100003"},{"_id":"55d66226726b611100aaf742","replac
ement":true,"quantity":183,"name":"Cooling
Fan","maintenanceperiod":0,"lifespan":0,"cost":300,"sku":"1
00004"},{"_id":"55d66226726b611100aaf743","replacement
":true,"quantity":143,"name":"Fuse
20A","maintenanceperiod":0,"lifespan":0,"cost":22,"sku":"10
0005"}]');
response.setStatusCode(200);
return response;
}
}
```

Challenge 7-Test scheduling logic

WarehouseSyncSchedule

```
global with sharing class WarehouseSyncSchedule
implements Schedulable{
global void execute(SchedulableContext ctx){
System.enqueueJob(new WarehouseCalloutService());
}
}
```

WarehouseSyncScheduleTest

```
@isTest
public class WarehouseSyncScheduleTest {
@isTest static void WarehousescheduleTest(){
String scheduleTime = '00 00 01 * * ?';
Test.startTest();
Test.setMock(HttpCalloutMock.class, new
WarehouseCalloutServiceMock());
String jobID=System.schedule('Warehouse
Time To Schedule to Test', scheduleTime, new
WarehouseSyncSchedule());
Test.stopTest();
//Contains schedule information for a
scheduled job. CronTrigger is similar to a cron job
on UNIX systems.
```

```
// This object is available in API version 17.0
and later.
CronTrigger a=[SELECT Id FROM CronTrigger
where NextFireTime > today];
System.assertEquals(jobID, a.Id,'Schedule ');
}
}
```